AMENDMENTS TO THE SPECIFICATION

Please delete the paragraphs beginning page 1, line 18 and continuing through line 25.

Please add the following paragraph on page 1, line 18:

CROSS-REFERENCE TO APPENDICES

Appendices A and B include an architectural specification and a programmer's guide. Appendix C includes a list of files contained on the computer program listing appendix (also submitted). Appendices A, B, and C are submitted on a compact disc. The compact disc containing Appendices A, B, and C hosts the following files: "A_Appndx.pdf," 7,589KB, "B_Appndx.pdf," 7,322KB, and "C_Appndx.pdf," 4,836KB. Additionally, a CD-ROM appendix containing a computer program listing is submitted on a compact disc. Each of the appendices are hereby incorporated herein by reference.

Please amend the paragraph beginning page 7, line 14 as follows:

The invention disclosed herein provides a mechanism for typing computer resources in a platform-independent distributed architecture. In particular, some embodiments of the present invention are described in detail in reference to the Hewlett-Packard E-Speak Service Engine Development Platform, Developer Release 3.03 (hereafter, "E-Speak Platform"). The E-Speak Platform is a distributed object infrastructure developed and released by Hewlett-Packard under an open-source ~~license~~license. ~~and freely available at "http://www.e-speak.hp.com."~~ The E-Speak Architectural Specification, Developer Release 3.03, and the E-Speak Programmer's Guide, Developer Release 3.03, both providing detailed specifications and descriptions of the E-Speak Platform, were also developed and released by Hewlett Packard under an open-source ~~license~~license.~~and are freely available at "http://www.e-speak.hp.com."~~ The E-Speak Architectural Specification, Developer Release 3.03, the E-Speak Programmer's Guide, Developer Release 3.03. and the E-Speak Service Engine Development Platform Release 3.03, are attached as CD-ROM Appendices A, B and ~~CD-ROM Appendix~~ C, respectively. It should be noted that although the present invention is herein disclosed primarily as implemented in the E-Speak Platform using object-oriented technology, it is not limited to any particular distributed platform or to any particular programming methodology, such as object-oriented programming. On the contrary, many distributed platforms may incorporate differing embodiments of the invention disclosed

herein, and, generally, any distributed computational process (e.g., a legacy application) may be adapted by conventional programming techniques e.g., wrapping the legacy application in a suitable object structure or otherwise adapting the process with a suitable interface-to employ the invention described herein. Additionally, a distributed architecture implementing an embodiment of the present invention may also be constructed using traditional procedural programming techniques (or a combination of object-oriented and procedural programming techniques).

Please amend the paragraph beginning page 9, line 21 as follows:

Figure 3 is a block diagram illustrating a process-level view of the distributed system of Figure 2. Computers 45 and 53 (or, more precisely, the address spaces of computers 45 and 53) execute Logical Machines 71 and 79 respectively. Logical Machines 71 and 79 comprise the infrastructure logic for the E-Speak Platform. The distributed system, in this simplified example, comprises infrastructure logic, i.e., Logical Machines 71 and 79, and the external resources --e.g., 43, 49, and 51-that they interconnect. Each Logical Machine 71 and 79 is a single instance of the E-Speak Platform. It should be noted that multiple Logical Machines may execute on a single physical machine, or the components of a single Logical Machine may be distributed across multiple machines. Each Logical Machine, e.g., 71 and 79, consists of a core, 75 and 77, and a repository, 78 and 83. Logical machines 71 and 79 are interconnected via their cores 75 and 77, and cores 75 and 77 are connected via communications medium 47. Each core, e.g., 75 and 77, consists of the process logic comprising the E-Speak Platform infrastructure; this includes. for example, process logic for registering, invoking, discovering and managing resources, for mediating and controlling resource access and access rights, for processing and routing messages between resources, and for maintaining client-specific interface data. The principal core processes are themselves core-managed resources. The core, e.g., 75, mediates all resource access (to both external and core-managed resources) using metadata stored in the repository, e.g., 78. Each resource, whether external or core-managed, which is accessible by a core has metadata describing it (vocabulary) and providing access to it (contract). The core, e.g., 75,[[89]] only operates on the resource metadata, and does not access resource implementation logic directly (except for core-managed resources). Instead, the core provides directs and controls communication between resources via message passing. A core "accesses" a resource specific implementation logic by passing a message to a resource-specific handler, or "resource

handler" (described in reference to Figures 4A and 4B); the resource handler, however, is not part of the core architecture.

Please amend the paragraph beginning page 10, line 25 as follows:

Figures 4A and 4B are block diagrams illustrating local and remote resource access in the distributed system depicted in Figure 2. In particular, resource access is illustrated in the simplified situation in which a client already possesses a reference to the resource, and thus the resource does not need to be discovered. Turning to Figure 4A, local resource access is illustrated in reference to a client connected to Logical Machine 71 accessing printer logic 72. Figure 4A shows logical machine 71, printer resource 72, resource handler 91, and client process 87 executing in the address space of machine 45. Client process 87 accesses the core 75 via a client library 89 providing APIs to the core 75; the client library 89[[83]] is provided as a component of the E-Speak Platform. Client process 87 is typically a resource requesting access to another resource, for example, a word processing resource requesting printing services from printing resource 72. Client 87 therefore sends a message to the core 75 naming the printer resource 72[[92]], e.g., by providing the URL for printer resource 72, and typically attaching a payload (in this case, for example, a document to be printed). After the core 75[[90]] receives the message, the core 75 accesses repository 78 for the metadata belonging to printer resource 72 to determine which resource handler processes requests to printer resource 72. The core 75 discovers the appropriate resource handler 91 from the metadata, and sends the message to it. The resource handler 91 then interfaces the resource implementation logic 72 and passes the service-dependent instructions and data to execute the client's 87 request. All local resource access is performed in the same manner. Thus, after printer resource 72[[92]] processes the request, it pass a result of the request back to the client process 87; in this situation, the printer resource 72[[92]] operates as a new "client process" sending a return message to the core, which is forwarded by the core to a resource handler for the client 87.

Please amend the paragraph beginning page 11, line 14 as follows:

Turning to Figure 4B, remote resource access is illustrated in reference to a client connected to Logical Machine 71 accessing remote printer/fax logic 85. In Figure 4B, core 75

and its repository 78, and core 77[[72]] and its repository 83, execute in different address spaces on different physical machines, e.g., computer 45 and computer 53 respectively. Cores 75 and 77 are connected via remote resource handlers associated with each core; thus, core 75 is connected to (and executes in the same address space as) remote resource handler 95, and core 77[[72]] is connected to (and executes in the same address space as) remote resource handler 97. The remote resource handlers 95[[99]] and 97 are, in turn, connected via communications medium 47. Client process 87, executing in the core 75 address space, sends an access request message to core 75[[87]] naming remote printer/fax resource 85, e.g., providing the URL to remote resource 85. A routing process in core 75 determines that remote resource 85 is local on core 77, and routes the message to remote resource handler 95. Remote resource handler 95 in turn sends the message to counter-part remote resource handler 97 executing in the core 77 address space. The remote resource handlers 95 and 97 mediate communication between cores 75 and 77, and maintain information such as which remote resource handler the message is to be routed, and the communication method or protocol to be used over communication medium 47. After the message is received by remote resource handler 97, the process of accessing remote printer/fax resource 85 is then identical to local resource access described in reference to Figure 4A; in this situation. however, remote resource handler 97 operates as the local client process sending the message to local core 77 (core 77 is now local to remote resource handler 97). Thus. Remote resource handler 97 sends the message to the core 77, which then accesses repository 83 for the metadata of printer/fax resource 85. The core 77 determines the appropriate resource handler 98[[105]]for the printer/fax resource 85, and sends the message to resource handler 98[[105]] accordingly. The resource handler 98[[105]]then interfaces printer/fax logic 85 and sends resource-specific data and instructions to printer/fax resource 85 to execute the request. Printer/fax resource 85 may then operate as a second client process when returning a message containing results back to client process 87.

Please amend the paragraph beginning page 12, line 21 as follows:

Figure 5A is a block diagram illustrating the resource typing mechanism of the present invention, according to some embodiments as implemented in the E-Speak Platform. Specifically, some embodiments of the resource typing mechanism are illustrated in Figure 5A in the context of registering an external resource with a core. Accordingly, a logical

machine 101, consisting of core 103 and repository 105, and external resource (implementation logic) 113 are shown in Figure 5A. External resource 113 is registered with the core using a core-managed resource. or resource factory 115. For purposes of registration, resource factory 115 must be provided with, among other items, a reference to the external resource (e.g., the implementation logic for the external resource), interface data for accessing the external resource, called the resource "contract" 111, an attribute list 107 conforming to a vocabulary 109 that describes the external resource 113, and the vocabulary 109 that defines the meaning of the attributes in the attribute list 107. The contract 111 or the vocabulary 109 used to register the external resource 113 may be pre-existing resources previously registered with the core 103, resources newly created by, e.g., the service provider for registering the particular external resource 113, or a core-supplied default contract or vocabulary resource. After receiving the registration request, the core-managed resource factory 115 types the external resource 113 by creating a metadata object 117 in repository 104[[105]], containing the attribute list data, and a reference to the vocabulary and contract data. In some embodiments, the external resource implementation logic may be included in the corresponding metadata items 117[[115]]; in other embodiment, only references, including a reference to the attribute list, may be included in the metadata items. In some embodiments, resource references may consist of URLs. In some embodiments, a resource may be registered and typed under multiple contract and vocabulary resources.

Please amend the paragraph beginning page 14, line 10 as follows:

Figure 5B is a block diagram illustrating the basic functional relationship between a resource, whether a service, vocabulary, or contract resource, and the vocabulary and contract resource with which it is typed, according to some embodiments. In general, resource 125 is advertised in vocabulary 121, and conforms to contract 123. In other words, resource 125 is described by the service provider with attributes describing the features (e.g., the purpose and functions) of the resource. The attributes comprising the attribute list ~~121~~, however, are provided individual meaning in reference to an explanatory schema, or vocabulary 121. The vocabulary 121 thus constitutes a schema by which to advertise the external resource 125 to other resources; other clients in possession of vocabulary 121 may thus decode the meaning of the attributes provided by the service provider in the description of the resource. Because a resource is typed with its vocabulary, the vocabulary is always available to provide meaning

to the descriptive attributes provided by the service provider. In addition, resource~~resources~~ 125 conforms~~conform~~ to its~~their~~ contract 123 because, as a definitional matter, the contract 123 specifies the interfaces supported by the external resource 125 (or in some cases, the contracts may contain stub classes for resource access). In some embodiments, a resource 125 may be typed with multiple vocabularies and multiple contracts. This is useful for resources such as printer/fax resource 85 in Figure 3. In the case of the printer/fax resource 85. for example, the printer/fax resource 85 may be typed advantageously with a first vocabulary describing solely printer attributes and a second vocabulary describing solely fax attributes, in addition to a third vocabulary describing printer-faxes. Thus, the printer/fax provider may advertise his printer/fax services to a client searching exclusively for printing services, and the client may never know that the printer services advertised also provide fax services.